

## 1.1 基于 OV5640 图像采集 SDRAM 缓存 TFT 显示系统设计及验证

|              |   |       |  |
|--------------|---|-------|--|
| 工程源码         | --altera 系列开发板标准配套资料<br> ---02_设计实例<br> ----- AC620_ov5640_sdram_tft<br> ----- AC620_ov5640_sdram_VGA<br> ----- AC620_ov5640_sdram_hdmi720p |       |  |
| 相关视频课程       | 暂无相关视频课程  |       |  |
| 本实验对各开发板支持情况 |   |       |  |
| 开发板型号        | AC108   | AC620 |  |
| 是否支持         | X   | √     |  |

如果您手头的硬件不支持本实验，您可以学习本实验的理论内容，也可以跳过本节内容，继续后续内容的学习。

### 本节导读

本节将讲述在 AC620 开发板上如何通过 OV5640 摄像头完成数据采集，并将采集到的数据使用 SDRAM 缓存后，满画幅显示到 800\*480 分辨率 TFT 显示屏上。

通过本节的内容，可以巩固学习前叙章节 OV5640 设计的理论基础、TFT、VGA 显示成像章节的学习内容，同时也是对上一章节 OV5640 图像采集 DPRAM 缓存 TFT 显示 160x128 图像的继续延伸。实现基于 FPGA 的摄像头数据采集输出设备显示是实现其数字图像处理的必经之路，后续的图像处理相关内容将以本实验内容作为理论基础，发挥 FPGA 数据处理实时性的优势，对采集到的数据作进一步分析处理。

### 1.1.1 系统原理

基于 OV5640 图像采集 TFT 或 VGA 显示系统设计，输入端为 OV5640 摄像头模块，输出端为 TFT 显示屏，FPGA 作为控制器和处理器，SDRAM 作为数据的缓存器，下图为本次系统设计的原理图：

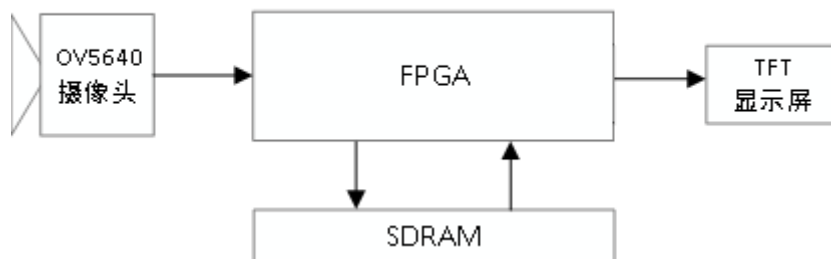


图 1.1-1 系统原理图

本节将基于 Altera Cyclone 系列 FPGA，结合 Omni Vision 公司的 1/4 英寸 CMOS QSXGA（2592x1944）图像传感器 OV5640，实现图像的采集与实时 TFT 满画幅显示。

经过分析，OV5640 这样一款高性能 500w 像素高清摄像头将采集到的图像发送给 FPGA。由于 FPGA 内部存储单元容量有限，FPGA 将接收到的数据输送到外接的大容量片外存储器 SDRAM 中进行缓存。缓存的图像数据经由 TFT 驱动模块读取并转换成 TFT 的输出格式，通过开发板的 GPIO 口输出以驱动 TFT 屏显示。

除了 TFT 格式输出外，通过 GPIO 口输出的数据还可以借助于 VGA 发送模块将图像数字信号发送到带 VGA 接收器的显示器，从而将其转换为模拟信号并驱动显示器显示。

由于 TFT 和 VGA 的输出数字信号格式完全相同，而仅仅是接口的形状存在差异，实际使用中，只需要在开发板上插接 GPIO-VGA 转换模块即可，为便于讲解，本节以分析 TFT 驱动信号生成的原理作为讲解重点。

那么，从总体架构来看，又为什么要设计 SDRAM 数据缓存器呢？这是因为如果寄希望于将图像数据存储在 FPGA 内部进行分析、比对、处理，则这对于 FPGA 的片上存储单元来讲，是一个很难完成的任务。上一章节已经分析了最大化利用 FPGA 片上存储单元得到输出图像的计算过程，从上一章的计算结果来看，要想让缓存的数据满画幅显示，就不得不寻找别的方法扩大缓存的区域，由于当前 SDRAM 容量普遍足够缓存一帧图像数据，所以使用 SDRAM 进行缓存便是考虑到的一种可行的解决方案。

设计好的缓冲区一次可以缓冲一帧 TFT 图像满画幅的数据，里面的数据会实时刷新，数据在读出的同时也在写入。这样，通过缓存的过程，摄像头采集到的数据和 TFT 屏显示数据的速度差异可以得到解决。由于某些特殊时刻 SDRAM 控制器会将一些读写数据忽略掉，所以在设计时还需解决这些时刻数据被忽略的问题，以保持数据接收的完整性。

## 1.1.2 总体设计及系统框图

理解了系统原理和设计思路后，接下来对 FPGA 内部模块功能作进一步划分和分析。

下图为本次基于 OV5640 图像采集显示系统的系统框图，相比于上一小节提供的原理图，本小节的系统框图给出了 FPGA 内部的模块搭建结构和数据流向：

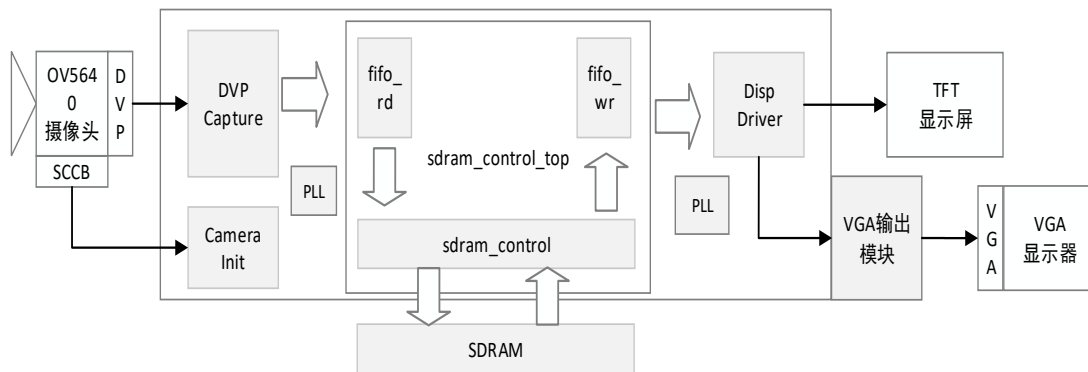


图 1.1-2 OV5640 图像采集 TFT/VGA 显示屏显示系统内部框图

整个系统最终的目的是要在 TFT 显示屏或者 VGA 显示器上显示图像数据，由于 TFT 显示屏采用的是 RGB565 数据格式，为了让图像正常的显示在显示屏的指定位置，需要有相应的时序发生逻辑，这就是显示屏驱动程序即上图中的 Disp\_Driver。该模块会按照 TFT 显示屏的接口时序产生对应的控制时序（HS、VS、DE、RGB data），实质上就是我们已经学习过的 TFT/VGA 控制器。

在前面的章节介绍过，OV5640 摄像头可以选择的图像数据输出格式有很多种，为了便于分析，我们选取了 RGB565 格式进行后续内容的讲解，这样可以同时保证和 TFT 显示屏的显示输出格式一致、数据位宽一致。使用 RGB565 格式进行数据采集和输出，每个像素的颜色数据使用 16 位数据即可表示，因此优先考虑缓存控制器的读出端口按照 16 位的数据位宽进行设计，由于缓存控制器的读出侧已经确定了读出数据位宽为 16 位，所以为了便于控制，可以设定缓存控制器的写入位宽也为 16 位，这样既可以兼顾摄像头的数据输出格式 RGB565，又可以保证输入和输出控制器的时钟频率相同。

前面我们也讲解了使用 2 端口带读写 FIFO 的 SDRAM 控制器设计原理和使用方法，这里，我们可以继续沿用这一模块组合，来缓存摄像头采集的数据。

经过以上分析，在上述系统中，2 端口的带读写 FIFO 的 SDRAM 控制器用来一并控制 FPGA 的片上读写 FIFO 和外部的 SDRAM 存储器，可以达到缓存需要显示的图像数据的目的。

由于缓存控制器输出侧的数据位宽为 16 位，这也就决定了我们需要将 DVP 接口接收的 8 位数据拼接为 16 位之后再写入，因此 DVP Capture 模块最主要的作用就是将 DVP 接口输出的连续的 8 位数据按照每 2 个数据一组拼接，得到 16 位数据。

据此，对于摄像头数据采集部分进一步总结，可以得出以下设计需求：

- 1、OV5640 要想能够正常的输出图像数据，必须经 SCCB 接口对其寄存器进行配置，所以整个系统首要的工作是使用控制逻辑经由 SCCB 接口对其进行初始化。完成这一任务的模块是 Camera\_Init，该模块会在系统开始工作时，对 OV5640 中各个寄存器写入指定值以实现初始化操作，其中典型如图像数据的输出格式 RGB565 就是通过其中一个寄存器的配置而指定的。
- 2、初始化完成后，摄像头采集到的图像数据按 8 位的输出格式，输出到 DVP Capture 模块。
- 3、DVP Capture 模块负责将 DVP 接口输出的数据按照每两个一组，得到符合 RGB565 图像格式的 16 位图像数据。
- 4、16 位图像数据由 sdram\_control\_top 控制器模块写入到 sdram 中。  
sdram\_control\_top 控制器内含 sdram 写入 fifo 和 sdram 读出 fifo，以此满足 sdram 和输入输出外设的工作速率差异和解决某些特殊时刻数据丢失的问题。

- 5、Disp\_Driver 模块从 sdram\_control\_top 模块读出数据，生成行场同步信号，通过 GPIO 口发送给外部 TFT 显示屏显示。
- 6、由于输入、缓存、输出的各模块对工作时钟的频率要求不一致，因此需要使用锁相环生成符合各个模块使用需求的时钟信号。

接下来，逐一介绍上述模块的设计过程。

### 1.1.3 模块设计

上面已经提出了模块的设计需求，即本次系统设计主要分为 5 个模块：

- 1、锁相环负责产生对应模块所需的时钟信号
- 2、摄像头初始化模块负责对摄像头进行初始化配置
- 3、数据流接收模块负责将摄像头采集到的数据以一定的格式输出到 FPGA
- 4、SDRAM 控制器模块组负责对 FPGA 接收的数据进行存储
- 5、TFT 屏显示驱动模块负责将 SDRAM 存储的数据以一定的时序输出并驱动 TFT 屏显示

各个模块详细说明介绍如下：

#### 1.1.3.1 锁相环设计

为了让后续开发工程的架构和当前工程保持一致，这里我们启用了两个锁相环模块，pll\_inst 和 pll\_hdmi。pll\_inst 主要负责 SDRAM\_control\_top 模块和摄像头初始化模块工作时钟的产生，pll\_hdmi 则主要负责显示模块时钟的产生。

关于 pll\_inst，该模块主要负责将系统的 50M 时钟倍频分频产生相应的时钟输送给对应模块，生成的相应时钟信号如下：

- C0: 该信号为 SDRAM 工作时钟信号, 该模块的工作时钟为 125MHz 系统时钟, 所以这里输出的时钟频率为 125MHz。
- C1: 该信号为一个向外输出的 125MHz 时钟, 时钟相位为-90 度。该信号在本工程中没有使用, 主要预留用于后期其他控制器接口, -90 度的时钟可以作为调整手段给更多的工程更灵活的时序余量选择。
- C2: 该时钟信号为输入时钟信号的输出同频同相位信号, 用于有 50MHz 时钟需求的驱动模块如摄像头初始化模块等。
- C3: 该信号为 OV5640 摄像头采集模块的工作时钟信号, 输出的时钟频率为 24MHz。由于 OV5640 本身自带有锁相环, 所以时钟会在 OV5640 摄像头采集模块内部再次倍频分频, 最后作为 DVP 的数字时钟信号输出。
- C4: 该信号为一个 33MHz 的工作时钟信号, 输送给 TFT 屏显示驱动模块 disp\_driver 驱动输出。

pll\_inst 的设置首页如下图所示:

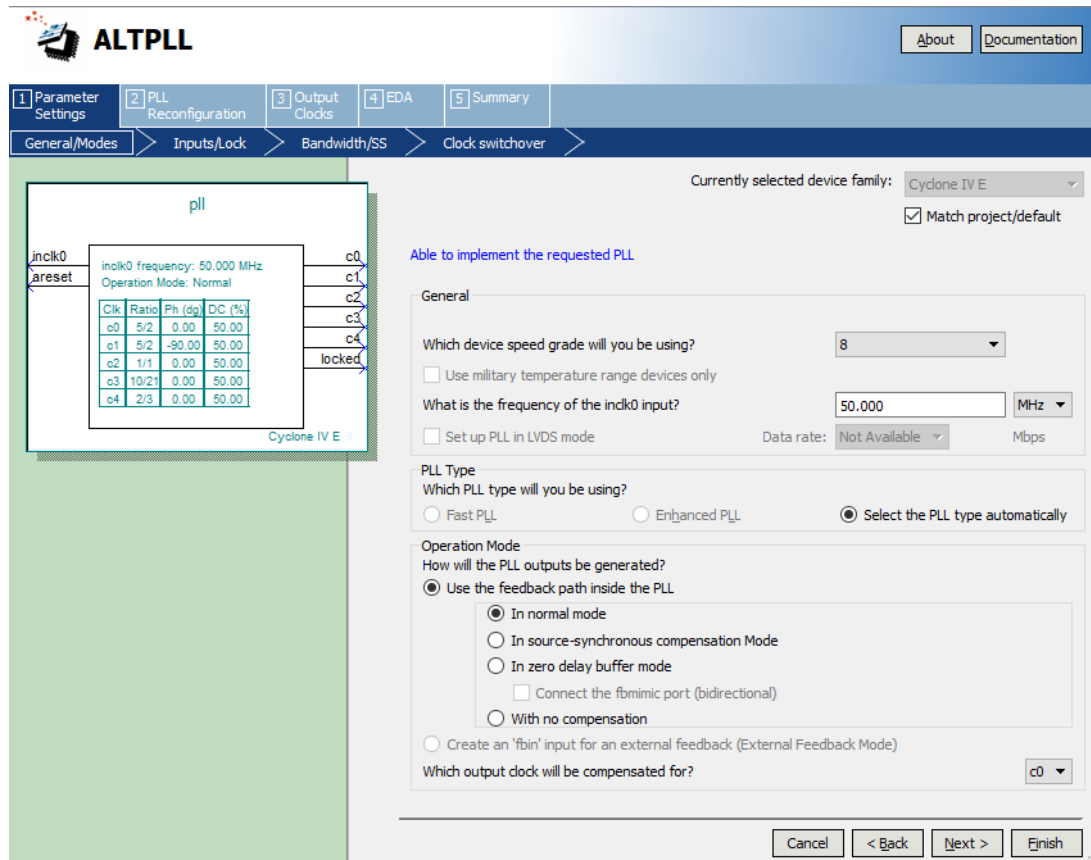


图 1.1-3 锁相环配置时钟输入配置窗口

由于每个锁相环可以配置 5 种输出时钟信号类别，因此，在本工程中，时钟信号的种类是刚好够用的，pll\_hdmi 的输出剩余接口，可预留给后续 hdmi 工程或其他对频率有不同需求的模块使用。

### 1.1.3.2 摄像头初始化模块（Camera\_Init）逻辑设计

所谓的 OV5640 摄像头初始化模块，其工作就是完成对 OV5640 中众多模式设置寄存器的写入操作。本质上就是使用 I2C 控制器将一些预先定义好的数据值写入到 OV5640 对应的寄存器中。所以我们使用一个 I2C 控制器加上一个存储好所有 OV5640 所需设置的寄存器参数的查找表，配合一定的控制逻辑实现，下图为我们设计的一个比较通用的 OV5640 初始化逻辑电路。



按照该思路，设计模块如图 1.1-4 所示

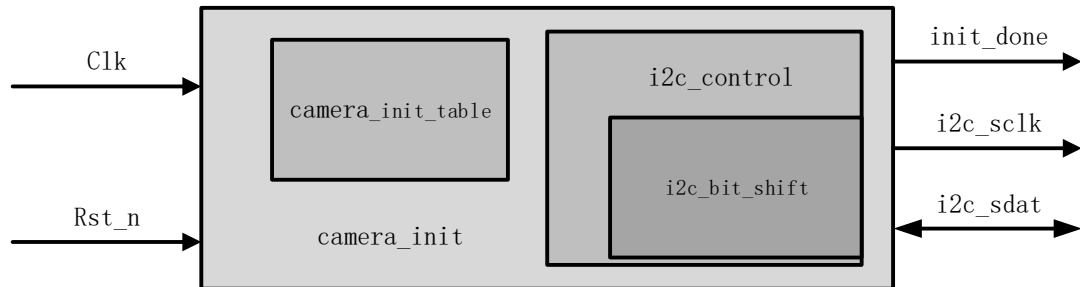


图 1.1-4 总体框架图

表 1.1-1 端口列表如下：

| 端口类型   | 端口名       | 描述                                |
|--------|-----------|-----------------------------------|
| Input  | Clk       | 系统时钟 50MHz                        |
| Input  | Rst_n     | 系统复位，低有效                          |
| Output | init_done | 摄像头初始化完成信号，如果摄像头完成初始化，则该引脚输出拉高信号。 |
| Output | i2c_sclk  | 摄像头初始化 SCCB（I2C）控制器的时钟总线          |
| Inout  | i2c_sdat  | 摄像头初始化 SCCB（I2C）控制器的数据总线          |

图中例化了设计好的通用 SCCB 控制器（i2c\_control）和一个 ROM 查找表（camera\_init\_table），在 camera\_init 顶层逻辑中，通过简单的状态机循环读取 camera\_init\_table 中的数据值并通过 i2c\_control 即可写入到 OV5640 的各个寄存器中。该代码已经提供在了我们的各个开发板配套资料包对应工程中。本模块经历过若干版本的升级，已经可以做到在以前的版本上精简控制信号的管脚，提高摄像头复位控制的可靠性，并对这些改进进行了充分的验证。

### 1.1.3.3 数据流接收模块（DVP\_capture）

OV5640 通过 DVP 时序接口对数据进行输出，根据 DVP 数据输出时序图，其输出时序参数如下图所示：

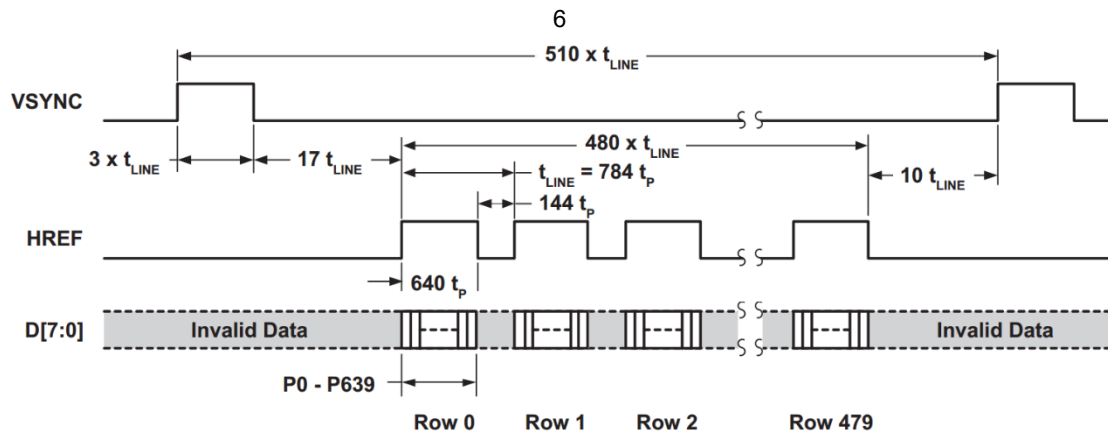


图 1.1-5 DVP 时序接口下输出图像数据的时序图

由上图可以看出，OV5640 在 DVP 时序接口下输出图像数据时，受两个控制信号控制，他们的具体作用如下：

- VSYNC，即帧同步信号。当设置 VSYNC 信号极性为高有效时，VSYNC 信号的高电平期间输出一帧图像。
- HREF，即行同步信号。当设置 HREF 信号极性为高有效时，高电平期间输出一行图像。

图像数据在 HREF 为高的时候输出，当 HREF 变高后，每一个 PCLK 时钟，输出一个 8 位/10 位数据。我们采用 8 位接口，所以每个 PCLK 输出 1 个字节，且在 RGB565/YUV422 输出格式下，每个  $t_p=2$  个  $T_{pclk}$ ，如果是 Raw 格式，则一个  $t_p=1$  个  $T_{pclk}$ 。比如我们采用 QSXGA 时序，RGB565 格式输出，每 2 个字节组成一个像素的颜色（低字节在前，高字节在后），这样每行输出总共有  $2592 \times 2$  个 PCLK 周期，输出  $2592 \times 2$  个字节。本节分析，主要以 RGB565 格式为主。

由于 DVP 接口输出的是 8 位数据，根据应用需求，整个 DVP\_Capture 模块的设计目的就是要实现每两个数据拼接为 1 个 16 位的数据并按照写 RAM 或 FIFO 的接口形式输出，模块设计框图如下图所示：

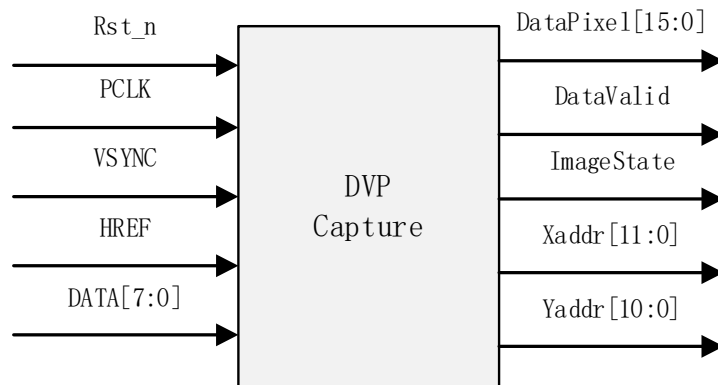


图 1.1-6 DVP Capture 数据流接收模块框图

OV5640 摄像头的输出端作为本模块输入端，这里可以结合如下时序图加深对这些信号的理解：

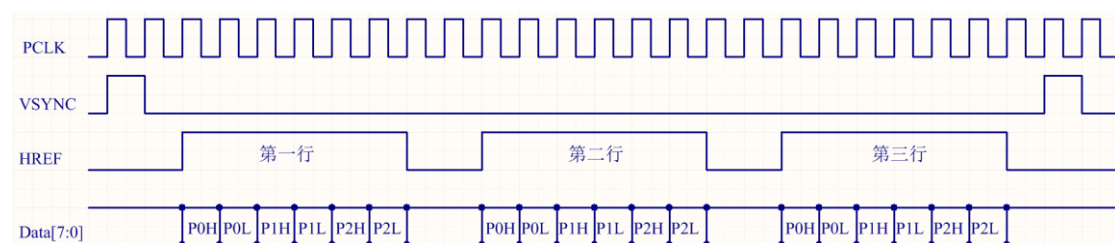


图 1.1-7 数据流接收模块输入信号时序关系示意图

上图中，每一帧图像开始前，VSYNC 会输出一个 PCLK 周期的高脉冲，随后，每行图像输出时，HREF 信号拉高，图像数据即开始输出。这里输出的数据格式，是按先高字节，后低字节绘制以作示意，而在实际使用中，可以通过配置摄像头初始化的寄存器值选择高低字节的输出顺序，以匹配缓存器对数据的缓存要求。

表 1.1-2 DVP 端口列表

| 端口类型 | 端口名 | 描述 |
|------|-----|----|
|------|-----|----|

店铺：<https://xiaomeige.taobao.com>

官方网站：[www.corecourse.cn](http://www.corecourse.cn)

技术博客：<http://www.cnblogs.com/xiaomeige/>

技术群组：

|        |             |   |
|--------|-------------|---|
| Input  | PCLK        | 像素时钟输出，根据输出模式和分辨率不同，该时钟频率值也不一样。   |
| Input  | Rst_n       | 系统复位，低有效  |
| Input  | VSYNC       | 帧同步信号。每幅图像开始输出前该信号会产生一个高脉冲，以用作帧同步功能   |
| Input  | HREF        | 行同步信号。该信号在 DATA 数据线输出一行图像数据的过程中一直保持高电平。   |
| Input  | DATA        | 8 位数据输出，数据的数据内容根据不同的模式设置有所不同，需要具体情况具体分析   |
| Output | DataPixel   | 两个 8 位数据合并为 16 位数据后输出的值   |
| Output | DataValid   | DataPixel 数据有效标志信号。稳定工作时一个像素时钟周期为高电平，一个像素时钟周期为低电平。                              |
| Output | ImageState  | 模块状态信号。当模块处于复位状态或未等到第一个场同步信号脉冲时，说明该模块未达到正常工作条件，输出高电平，当第一个场同步信号脉冲到来时，本信号由高电平拉低输出 |
| Output | Xaddr[11:0] | 行像素坐标，使用计数器实现   |
| Output | Yaddr[10:0] | 列像素坐标   |

模块中，DataPixel 为 16 位的 RGB565 格式的像素数据，由连续的 2 个 Data 数据拼接而来。DataValid 为 DataPixel 数据有效标志信号，由于 DATA 端口需要 2 个时钟才能传输一个像素所需的 16 位数据，所以 DataPixel 端口上的数据理论来说应该是每 2 个时钟周期只有一个时钟周期是真正有效的，所以 DataValid 在连续的两个时钟周期中，只有一个时钟周期为高电平，一个时钟周期为低电平，来确保下一级在使用 DataPixel 时，每两个时钟周期内只使用一次。在将数据直接写入 FIFO 时，可以直接将 DataValid 信号当做 wrreq 信号使用。

在有些应用中，需要实时知道当前输出的图像数据在输出像素矩阵中的绝对位置，以便于根据不同的位置进行不同的处理。最典型的如使用图像处理技术定位图像中某个点的具体位置。则需要用到 Xaddr 和 Yaddr，因此，该 DVP Capture 模块也将每个像素对应的位置通过 Xaddr 和 Yaddr 两个端口输出。

需要关注的是，在模块中，Xaddr 和 Yaddr 都是从 1 开始算的，并非从 0 开始，也就是说，输出的每行最开头的一个像素对应的 Xaddr 的值为 1，输出的每帧图像的第一行数据，其 Yaddr 的值也为 1。

### 1.1.3.4 SDRAM 控制器模块组 (SDRAM\_control\_top)

该控制器模块和读写两个 FIFO 共同组成了 SDRAM\_control\_top 模块，其中写 FIFO 负责将 OV5640 采集的数据缓存，由 SDRAM 控制器读取并控制其存储到片外存储设备 SDRAM 中，读 FIFO 负责对 SDRAM 中要读出的数据缓存并在 SDRAM 控制器的控制下，当 TFT 显示驱动模块需要数据时，将其传输给 TFT 显示驱动模块，为 SDRAM 控制器加入两个 FIFO，该设计很好地解决了在某些特殊的时刻，有些读或写会被忽略掉，而且数据的写或读不能连续对数据流进行缓存，只能间歇式的读或写 SDRAM 数据，导致数据存储或读取遗漏的问题。关于 SDRAM 控制器和两 FIFO 之间的关系框图如下：

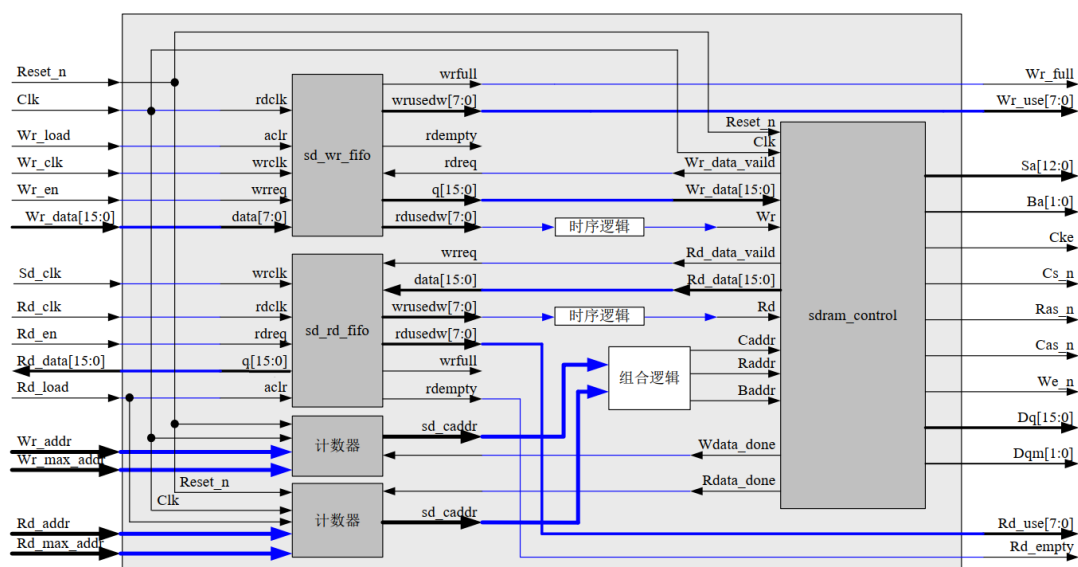


图 1.1-8 sdram\_control\_top 模块组的结构框图

表 1.1-3 sdram\_control\_top 模块组的端口名称列表

| 端口类型 | 端口名 | 描述 |
|------|-----|----|
|------|-----|----|

店铺: <https://xiaomeige.taobao.com>

官方网站: [www.corecourse.cn](http://www.corecourse.cn)

技术博客: <http://www.cnblogs.com/xiaomeige/>

技术群组:

|        |             |  |
|--------|-------------|--|
| Input  | Clk         | SDRAM_control_top 模块组的工作时钟，兼作 wr_fifo 的读时钟   |
| Input  | Reset_n     | SDRAM_control_top 模块组的复位信号   |
| Input  | Wr_load     | wr_fifo 的数据加载启动信号。当需要存储的数据还没有准备好时，将其拉高使 wr_fifo 一直处于复位状态，当希望 sdram_control_top 模块组缓存的数据到来时，将其拉低。 |
| Input  | Wr_clk      | wr_fifo 的写时钟   |
| Input  | Wr_en       | wr_fifo 的写使能信号   |
| Input  | Wr_data     | wr_fifo 的写入数据  |
| Input  | Sd_clk      | rd_fifo 的写信号时钟   |
| Input  | Rd_clk      | rd_fifo 的读信号时钟   |
| Input  | Rd_en       | rd_fifo 的读使能信号   |
| Input  | Rd_data     | rd_fifo 读出的数据  |
| Input  | Rd_load     | rd_fifo 的数据加载启动信号。当需要存储的数据还没有准备好时，将其拉高使 wr_fifo 一直处于复位状态，当希望 sdram_control_top 模块组缓存的数据到来时，将其拉低。 |
| Input  | Wr_addr     | 写入 sdram 的写入起始地址，占用 sdram 的存储区域起始地址  |
| Input  | Wr_max_addr | 写入 sdram 的写入最大地址，占用 sdram 的存储区域最大地址  |
| Input  | Rd_addr     | 读出 sdram 的读出起始地址，读出区域的起始地址，可以和写入占用 sdram 的存储区域起始地址相同   |
| Input  | Rd_max_addr | 读出 sdram 的读出最大地址，读出区域的最大地址，可以和写入占用 sdram 的存储区域最大地址相同   |
| Output | Wr_full     | wr_fifo 满信号  |
| Output | Wr_use      | wr_fifo 内的数据存量   |
| Output | Sa          | （直连板载 SDRAM）地址总线。结合命令有不同意义   |
| Output | Ba          | （直连板载 SDRAM）Bank 地址。用于选择不同的 Bank 进行读写操作  |
| Output | Cke         | （直连板载 SDRAM）时钟使能。屏蔽系统时钟，来冻结当前操作，当该引脚为高电平时，所有引脚电平才能被正确送入 SDRAM 芯片                                 |
| Output | Cs_n        | （直连板载 SDRAM）片选信号。用于屏蔽和使能所有输入端口，但不包括 CLK、CKE 和 DQM，低电平有效  |
| Output | Ras_n       | （直连板载 SDRAM）行地址选通。该信号为低时，在时钟的上升沿锁存行地址，使能行访问和预充电  |
| Output | Cas_n       | （直连板载 SDRAM）列地址选通。该信号为低时，在时钟的上升沿锁存列地址，使能列访问  |
| Output | We_n        | （直连板载 SDRAM）写使能信号，该信号为低时，使能写操作和预充电   |
| Output | Dq          | （直连板载 SDRAM）数据总线，数据输入输出复用  |
| Output | Dqm         | （直连板载 SDRAM）数据掩码。L(U)DQM，低（高）字节掩码，当其为高时，下一时钟的上升沿时数据总线的低（高）字节为高阻态                                 |
| Output | Rd_use      | rd_fifo 数据存量   |
| Output | Rd_empty    | rd_fifo 读空标志位，当 rd_fifo 被读空时拉高   |

为了分析上述表格中的信号，可以将信号归类分成 3 个组来分析：

店铺：<https://xiaomeige.taobao.com>

官方网站：[www.corecourse.cn](http://www.corecourse.cn)

技术博客：<http://www.cnblogs.com/xiaomeige/>

技术群组：

组 1：上述表格中标注了（直连板载 SDRAM）字样的信号。这些信号在使用本模块组的时候不必过多关注配置方案，只需要将其作为 FPGA 端口和 SDRAM 硬件作好对接即可。这些信号是 SDRAM 控制器和 SDRAM 存储器的交互信号，而在源码设计中，模块组只面向读写 fifo 进行操作，面向 SDRAM 存储器的控制是由读写 fifo 进一步完成，所以源码设计面向 SDRAM 控制器和存储器是透明的，只需要对读写 fifo 负责即可。

组 2：Wr\_信号名开头的信号。这一组信号负责将送来的数据进行写 fifo 缓存，然后转交给 SDRAM 进行再一次的缓存。

组 3：Rd\_信号名开头的信号。这一组信号负责将 SDRAM 的缓存数据读出，然后再 fifo 缓存后输出。

此外，sdram\_control\_top 模块组内部还有互联信号，互联信号主要是负责读写 fifo 和 SDRAM 进行数据交互时的信号。

SDRAM 要保证正常上电后可以开始数据的读写，需有初始化的过程。这一过程主要涉及到读、写、刷新等过程及其状态跳转的相互关系。而这一过程，需借助 sdram\_control\_top 模块组内部 SDRAM 控制器（SDRAM\_control）完成。

SDRAM 控制器（SDRAM\_control）的作用是负责控制 SDRAM 中数据的读取和刷新，整个 SDRAM 控制器采用状态机实现，状态机的状态转移图如下图所示。



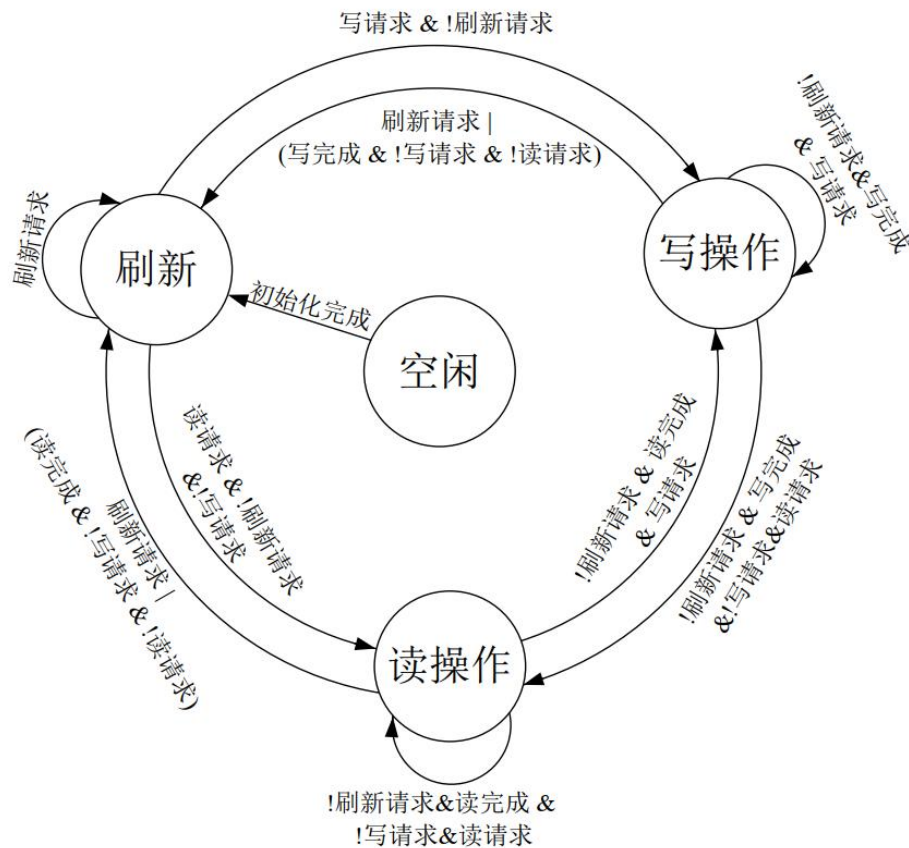


图 1.1-9 SDRAM 初始化过程状态转移图

SDRAM 在上电后，系统处于空闲状态，在上电初始化完成后进入刷新状态，此状态主要负责自动刷新操作，在此状态如果有读或写请求到来，状态将会转移到相应的读或写状态。读或写状态各自主要负责读写数据操作。整个状态机中对刷新操作、写数据操作、读数据操作是有一个优先级的，也就是某两个或多个操作请求同时到来时，先执行哪个操作，这里默认的优先级是刷新操作>写数据操作>读数据操作。数据在写入和读出 SDRAM 时，其默认起始地址都为 0，终止地址则由初始化时设定的图像输出宽度 IMAGE\_WIDTH 与图像输出高度 IMAGE\_HEIGHT 的乘积所决定，例如这里我们设定的是宽为 800，高为 480，所以在读写数据时其终止位为 384,000。



```
parameter IMAGE_WIDTH  = 800; //图片宽度
parameter IMAGE_HEIGHT = 480; //图片高度(≤720)
parameter IMAGE_FLIP    = 0;   //0: 不翻转, 1: 上下翻转
parameter IMAGE_MIRROR  = 0;   //0: 不镜像, 1: 左右镜像
```

图 1.1-10 图片参数源码

关于上电初始化的内容和相关引脚的更具体讲解，可以参考前述文章中的专门内容，这里就不作进一步分析。

由于上述模块组端口列表中划分的组 1 主要和初始化过程密切相关，并且属于底层硬件通信过程，所以实际操作时可以不关心其具体实现过程。剥离了组 1 并分析了 SDRAM 初始化过程后，SDRAM 的存储控制信号理解起来和 FIFO 的存储控制信号是极其相似的。下面讲解 FIFO 的配置过程。

注意由于数据在写入和读出 FIFO 时都为 16 位，所以在设置 FIFO 时要将其位宽设置为 16 位，由于数据的写入和读出是同时进行的，所以这里 FIFO 设置为双时钟，数据深度这里设置为 512，由于读写 FIFO 的配置相似，这里以读 FIFO 的配置为例进行分析，具体设置如图所示：

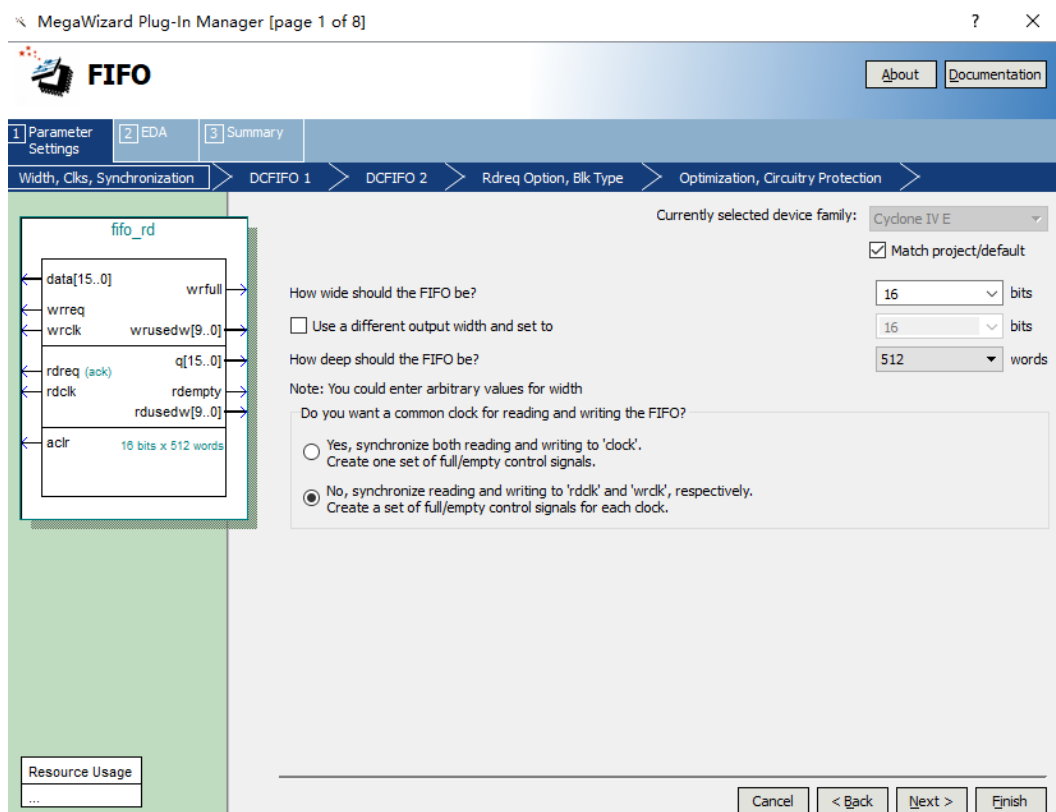


图 1.1-11 读 fifo 设置图 1

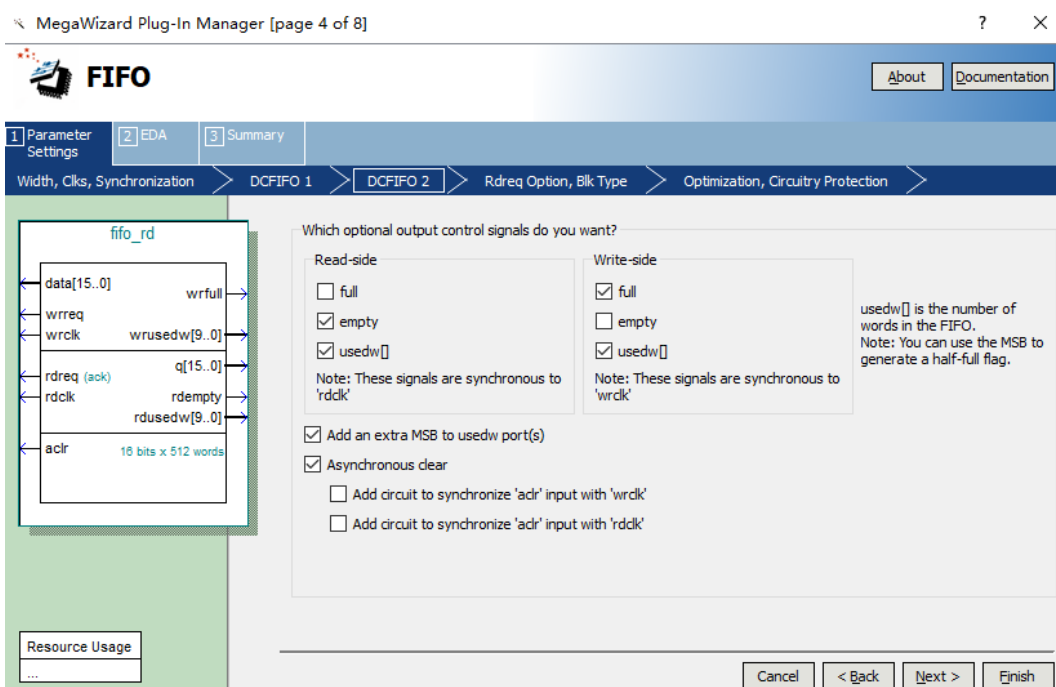


图 1.1-12 读 fifo 设置图 2

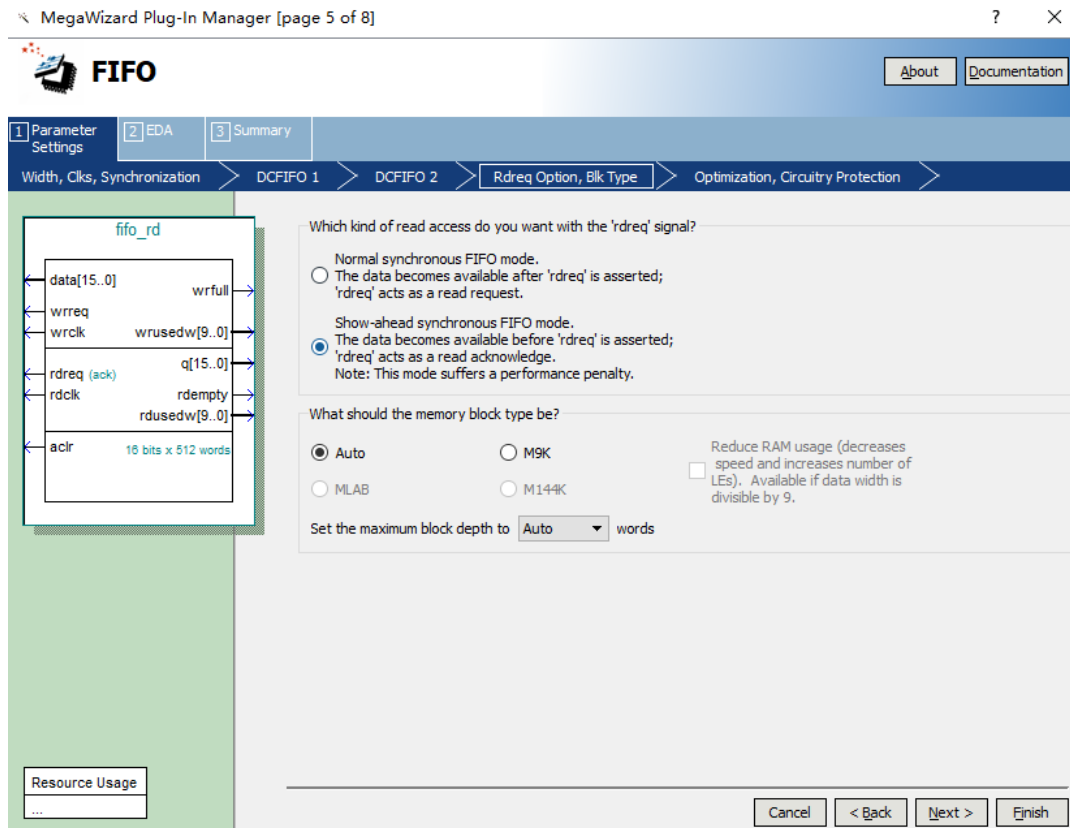


图 1.1-13 读 fifo 设置图 3

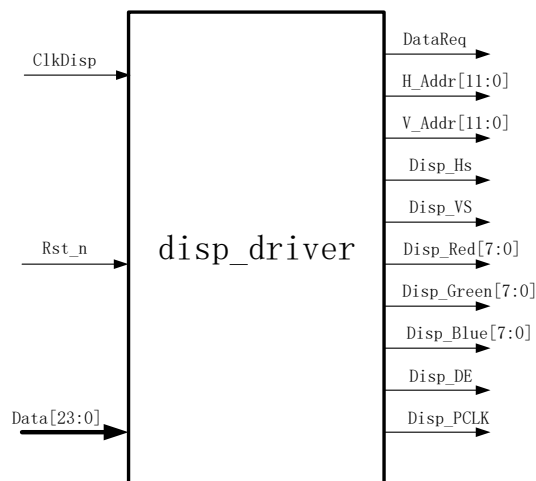
分析完 SDRAM 控制器模块组的对外接口后，简单分析一下模块组内部的两个关键的互联信号：图 1.1-8 中，sram\_control 模块（注意，这里是模块组内部的子模块）的写使能 Wr 和读使能 Rd 是分别通过判断写 FIFO 模块和读 FIFO 模块中所存储的数据量来决定的。在写 FIFO 模块中，当存放的数据量大于一次突发写长度数据量时就将写使能 Wr 拉高；在读 FIFO 模块中，当存放的数据量小于整个 FIFO 能存放数据量的一半时就读使能 Rd 拉高。这样就能解决上述在连续数据流读取时，数据读取存在遗漏的问题。

sram\_control\_top 模块组的实现虽然分析起来比较复杂，但是实际在工程应用中，只需要学会如何例化使用它即可。而使用它的核心内容，是对其读写 FIFO 的操作。后面在顶层模块设计分析的内容中，将讲解如何例化和使用它。

### 1.1.3.5 TFT 屏显示驱动模块（disp\_driver）

正如在前面系统框图中介绍的一样，该模块主要负责驱动 TFT 屏的显示，通过使用两个计数器分别进行行、场计数，然后根据计数器的计数值确定像素数据内容和行、场同步信号的电平状态，进而生成相应的控制时序，数据在输送给 TFT 显示驱动模块时为 16 位的 RGB565 格式。

为了提高模块的兼容性，模块输入输出按 RGB888 设计以提供足够的位宽输入，如果接收到的数据为 RGB565 格式显示颜色，可以通过各颜色分量的低位补 0 凑齐为 RGB888 格式。同样的道理，输出解析的各颜色分量，也分别按高位取相应的位宽作为 RGB565 输出即可。如下图所示：



| 端口类型   | 端口名     | 描述   |
|--------|---------|--|
| Input  | ClkDisp | 模块工作驱动时钟，和需要外接的输出设备工作频率相关  |
| Input  | Rst_n   | 模块复位信号   |
| Input  | Data    | 输入模块的像素颜色数据，这里为提高兼容性按 RGB888 设计，如果只需要 RGB565 格式显示颜色，可以通过各颜色分量的低位补 0 凑齐为 RGB888 格式。 |
| Output | DataReq | 屏幕可见显示区标识信号，可用于作为向其他模块请求数据的标识信号  |
| Output | H_Addr  | 图像区行扫描地址   |
| Output | V_Addr  | 图像区场扫描地址   |
| Output | Disp_HS | 行同步信号  |

|        |            |   |
|--------|------------|---|
| Output | Disp_VS    | 场同步信号   |
| Output | Disp_Red   | 输出像素值的 8 位红色分量，如果用 RGB565 格式，则输出后取位宽为[7:3]位即可 |
| Output | Disp_Green | 输出像素值的 8 位绿色分量，如果用 RGB565 格式，则输出后取位宽为[7:2]位即可 |
| Output | Disp_Blue  | 输出像素值的 8 位蓝色分量，如果用 RGB565 格式，则输出后取位宽为[7:3]位即可 |
| Output | Disp_DE    | 数据发送使能，用于控制图像的显示区域。                           |
| Output | Disp_PCLK  | 外部显示设备工作的驱动时钟                                 |

上面的表格，对本模块的各输入输出接口引脚作了简单的功能描述。

该模块产生的时序控制（HS、VS、DE、RGB DATA），其中 HS 为 TFT 的行同步信号，VS 为 TFT 的场同步信号，DE 为背光使能，RGB data 为颜色数据，这里我们将其拆分为 Disp\_Red，Disp\_Green，Disp\_Blue 输出。下表为 RGB565 模式下三种颜色对应的数据位。

| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| R4  | R3  | R2  | R1  | R0  | G5  | G4 | G3 | G2 | G1 | G0 | B4 | B3 | B2 | B1 | B0 |

由于 AC620 开发板带有 TFT 接口，所以数据在经过该模块驱动输出到 TFT 屏后，用户就已经可以通过 TFT 屏观察到图像了，当然也可以在 GPIO 口处插接 VGA 模块，通过 VGA 显示器观察输出结果。实现 VGA 显示的过程在通过 GPIO 口输出到外部后，更多一步数模转换和模数转换的过程，即通过 GPIO 口输出的视频数字信号经过数模转换后，转换完成的模拟信号经由 VGA 接口传输到 VGA 显示器上，再经由 VGA 显示器上相应的驱动程序驱动，输出在 VGA 显示屏上。

本模块是为驱动 TFT 屏而定制，所以要想更好的理解本模块的设计思想，可以查阅配套的 TFT 屏的设计参考手册加以更深层次的理解。

## 1.1.4 顶层模块设计分析

完成了各子模块的设计后，接下来进行顶层模块的设计分析。为了实现各模块接口的信号连接，接下来进行顶层模块的设计分析讲解。在顶层只进行接口定义、模块例化和简单的连线处理可以保证良好的代码可读性，同时也是良好架构设计风格的体现。

## 1.1.5 锁相环的例化

在本工程中，使用了两个锁相环模块。由于每一个锁相环支持 1 路时钟输入，5 路时钟输出，经过分析，如果只需要满足本工程的实验要求，将 clk\_vga 信号使用第一个锁相环生成，实际上也是可行的。例化第二个锁相环，可以保证工程在未来进行 HDMI 设计时，能获得 HDMI 需求的工作时钟。

```
pll pll_inst (  
    .inclk0(clk),  
    .areset(!rst_n),  
    .locked(locked),  
    .c0(clk_sdr_ctrl),  
    .c1(sdram_clk),  
    .c2(clk_50m),  
    .c3(clk_24m)  
);
```

```
pll_hdmi pll_hdmi_inst(  
    .inclk0(clk),  
    .c0(clk_vga),  
    .c1(clk_vgax5)  
);
```

介绍完锁相环的例化过程，接下来介绍摄像头工作时钟的生成。

### 1.1.5.1 摄像头的工作时钟

```
assign camera_xclk = clk_24m;
```

摄像头工作时钟的生成实际上是由锁相环生成的 24M 时钟直接连线引出，向外提供给摄像头模块使用。

### 1.1.5.2 参数设置

```
localparam RGB = 0;
localparam JPEG = 1;

parameter IMAGE_WIDTH = 800; //图片宽度
parameter IMAGE_HEIGHT = 480; //图片高度(≤720)
parameter IMAGE_FLIP = 0; //0: 不翻转, 1: 上下翻转
parameter IMAGE_MIRROR = 0; //0: 不镜像, 1: 左右镜像
```

这里可以设置图片的分辨率和数据格式，还可以设置图像的镜像和翻转。

### 1.1.5.3 摄像头初始化例化

```
camera_init
#(
    .CAMERA_TYPE ( "ov5640" ),// "ov5640"
    .IMAGE_TYPE ( 0 ),// 0: RGB; 1: JPEG
    .IMAGE_WIDTH ( IMAGE_WIDTH ),// 图片宽度
    .IMAGE_HEIGHT ( IMAGE_HEIGHT ),// 图片高度
    .IMAGE_FLIP_EN ( 0 ),// 0: 不翻转, 1: 上下翻转
    .IMAGE_MIRROR_EN ( 0 ) // 0: 不镜像, 1: 左右镜像
)camera_init
(
    .Clk (clk_50m ),
    .Rst_n (locked ),
    .Init_Done (Init_Done ),
    .camera_rst_n( ),
    .camera_pwdn ( ),
    .i2c_sclk (camera_sclk ),
    .i2c_sdat (camera_sdat )
);
```

在摄像头初始化例化代码中，可以将顶层设置的参数（包括摄像头型号，图像宽度、图像高度，图像的镜像和翻转）传递到子模块。当这些参数传递到

子模块后，如果子模块也有相同名称的参数并进行了赋值，则以上层定义为准，子模块的参数自动失效。

摄像头初始化控制器的驱动时钟选用 50M 时钟即可。由于只有锁相环工作稳定后，摄像头初始化的时钟才能保证是稳定的时钟，所以用锁相环的 locked 信号对应摄像头初始化控制器的复位信号。当锁相环还未稳定时，locked 一直为低电平，即 camera\_init 模块一直未开始工作，直至等到锁相环复位完成将 locked 拉高，继而开始摄像头初始化的工作。

有的读者没有注意模块工作的先后要求，直接盲目的将工程复位信号和摄像头初始化复位信号对应，这样是考虑不周的，究其原因是锁相环提供稳定的频率是需要时间的。提前启动摄像头初始化模块，最直接的后果是有可能导致摄像头的配置寄存器写入失败。

#### 1.1.5.4 数据流接收模块例化

```
DVP_Capture DVP_Capture(  
    .Rst_n(Init_Done),  
    .PCLK(camera_pclk),  
    .Vsync(camera_vsync),  
    .Href(camera_href),  
    .Data(camera_data),  
    .ImageState(fifo_aclr),  
    .DataValid(fifo_wrreq),  
    .DataPixel(fifo_wrdata),  
    .Xaddr(),  
    .Yaddr()  
);
```

如果摄像头初始化的步骤没有完成，则 Init\_Done 信号为低电平，只有当摄像头初始化完成后，摄像头初始化模块发出的 Init\_Done 信号才会被拉高，接入到本模块中。Init\_Done 拉高接入到本模块后，Rst\_n 信号从低电平转为高电平，本模块开始工作。



本模块的 PCLK 信号对应的是 camera\_pclk 信号，实际上是将摄像头模块提供的时钟接入本模块之中，在保证时钟频率正确的前提下，这样可以保证时钟和数据同源，时钟和数据同源可以确保时钟的节拍和数据的节拍有良好的时序关系。

ImageState 向写 fifo 告知本模块的当前状态，只有当其拉低，才能通过写 fifo 的复位信号控制有效数据的写入，否则表明数据流前端的数据还没准备好，数据管脚上的数据是没有意义的，阻止写 fifo 写入数据。

DataValid 和 DataPixel 这一组信号，精确的向数据流下游，即写 fifo，提供 16 位写入的数据及符合时序的数据生效信号。

从更深层次理解数据收发的过程来看，Rst\_n、PCLK、ImageState 和 DataValid，都是为像素数据的精确控制而服务的。Rst\_n 解决的是模块层面的 8 位数据接收的使能问题，PCLK 即数据接收时钟，解决的是节拍层面的 8 位数据接收使能问题，ImageState 解决的是模块层面的 16 位数据发送使能问题，DataValid 解决的是时钟层面的 16 位数据发送使能问题。

### 1.1.5.5 SDRAM 控制器模块组例化

```
sdr_control_top sdr_control_top(  
    .Clk(clk_sdr_ctrl),  
    .Rst_n(rst_n),  
  
    //wr_fifo interface  
    .Wr_data(fifo_wrdata),  
    .Wr_en(fifo_wrreq),  
    .Wr_addr(0),  
    .Wr_max_addr(IMAGE_WIDTH*IMAGE_HEIGHT),  
    .Wr_load(!Init_Done),  
    .Wr_clk(camera_pclk),  
    .Wr_full(),  
    .Wr_use(),  
);
```

```
//rd_fifo_interface
.Rd_data(RGB_DATA),
.Rd_en(DataReq),
.Rd_addr(0),
.Rd_max_addr(IMAGE_WIDTH*IMAGE_HEIGHT),
.Rd_load(!Init_Done),
.Rd_clk(clk_vga),
.Rd_empty(),
.Rd_use(),

//sdram_interface
.Sa(sdram_addr),
.Ba(sdram_ba),
.Cs_n(sdram_cs_n),
.Cke(sdram_cke),
.Ras_n(sdram_ras_n),
.Cas_n(sdram_cas_n),
.We_n(sdram_we_n),
.Dq(sdram_dq),
.Dqm(sdram_dqm)

);
```

从 SDRAM 的例化接口对应关系来看，值得特别关注的地方在于 wr\_fifo 的启动写入时机和 rd\_fifo 的启动读出时机。由于从摄像头捕获的真实有效的数据至少要等到摄像头初始化完成后才能确保摄像头输出数据有意义，所以使用摄像头初始化完成的信号（Init\_Done）作为 wr\_fifo 和 rd\_fifo 的工作启动信号。当摄像头初始化完成后，Init\_Done 信号拉高，取反后和 Wr\_load 及 Rd\_load 对应。

在 SDRAM 二端口控制器模块组内部，实际上 Wr\_load 和 Rd\_load 直接连接对应的分别是两个 FIFO 的清零信号 aclr。这样，当摄像头初始化未完成，两个 FIFO 始终处于清零状态，当摄像头初始化完成，则清零释放，两个 FIFO 可以开始接收数据。由于其他模块面向本模块组的操作是透明的，所以读写 FIFO 可以开始接收数据视作整个 sdram\_control\_top 模块组可以开始缓存数据。

纵观本模块组的时钟信号和控制信号设计，可以从如下角度进行理解：

- 1、模块组引入 Rst\_n 信号，是作为整个模块组的复位信号，这个信号和顶层全局复位控制是保持一致的，模块组上电即释放复位，开始生效。
- 2、虽然模块组解除 Rst\_n 复位而开始正常工作，但是这时候，读写 FIFO 还未允许工作，并且内部数据未知不可控。所以此时必须等待摄像头产生的有效数据输出，否则写入两个 FIFO 的数据都是错误和未知的。
- 3、Init\_Done 信号的拉高，表明了摄像头有效数据输出。此时，读写 FIFO 允许开始接收数据。
- 4、wr\_data 和 wr\_en 配合控制数据正确写入 wrfifo。两组信号由数据流上游 DVP\_Capture 模块而来，上游的有效数据被送到数据总线，有效数据的输出和使能拉高是同步的，以此确保写 FIFO 接收数据就是正确有效的。
- 5、写 FIFO 的时钟为什么为 camera\_pclk 呢？因为写 FIFO 的 wr\_clk，即写 FIFO 的工作时钟，和摄像头的输出像素时钟保持一致即可和输入 FIFO 的数据同步，每一拍向 FIFO 中写入一个像素数据。
- 6、读 FIFO 的时钟由 TFT/VGA 的输出频率而决定并保持一致。
- 7、读使能 Rd\_en 和下游 DataReq 连接，当下游 DataReq 拉高时，读 fifo 的 Rd\_en 被拉高，此时数据 RGB\_DATA 按 clk\_vga 的时钟节拍，源源不断流出，送往下游 disp\_driver 模块。而前面提到过，下游的 DataReq 功能，实际上是进行的像素有效区域选择。

### 1.1.5.6 TFT 屏显示驱动模块例化

```
disp_driver disp_driver(  
    .ClkDisp      (clk_vga      ),
```

店铺：<https://xiaomeige.taobao.com>

官方网站：[www.corecourse.cn](http://www.corecourse.cn)

技术博客：<http://www.cnblogs.com/xiaomeige/>

技术群组：

```
.Rst_n      (rst_n      ),
.Data       ({RGB_DATA[15:11],3'd0,RGB_DATA[10:5],2'd0,RGB
_DATA[4:0],3'd0}),
.DataReq    (DataReq    ),
.H_Addr     (H_Addr     ),
.V_Addr     (V_Addr     ),
.Disp_HS    (video_hs   ),
.Disp_VS    (video_vs   ),
.Disp_Red   (video_r    ),
.Disp_Green (video_g    ),
.Disp_Blue  (video_b    ),
.Disp_DE    (video_de   ),
.Disp_PCLK  (video_clk  )
);
```

在这个模块之中，驱动时钟选择的是 clk\_vga,这样可以保证在数据处理时钟频率和上游 SDRAM 缓存模块组时钟频率一致，复位选择全局复位即可。

前面我们介绍过，由于 disp\_driver 模块在设计时采用了兼容 RGB888 模式和 RGB565 模式，所以本模块的 Data 数据输入接口设计为 24 位。为了解决 RGB888 和 RGB565 的数据格式兼容问题，Data 数据输入前，必须要将 RGB565 数据进行三元色低位拼接补 0 的预处理。

### 1.1.5.7 像素输出数据的处理

```
assign tft_rgb = {video_r[7:3], video_g[7:2], video_b[7:3]}; //TFT
数据输出
```

tft\_rgb 是输入数据 Data 在像素有效区域内相同格式即 RGB565 的显示数据，所以，在得到位宽为 8 的三元色 video\_r/video\_g/video\_b 的像素数据后，需将其 RGB888 格式转换回 RGB565 格式。转换的方式为按三元色选取 RGB888 高位位宽连线对应的值分别作为 RGB565 的各三元色像素颜色值。

### 1.1.5.8 综合完成后的 RTL 模块框图

经过上述各子模块的设计和顶层模块整合设计，基于 OV5640 图像采集 SDRAM 缓存 TFT 设计显示输出的设计内容已经全部设计完成。从软件中查看 RTL 图，可以看到设计与预期是相符的。

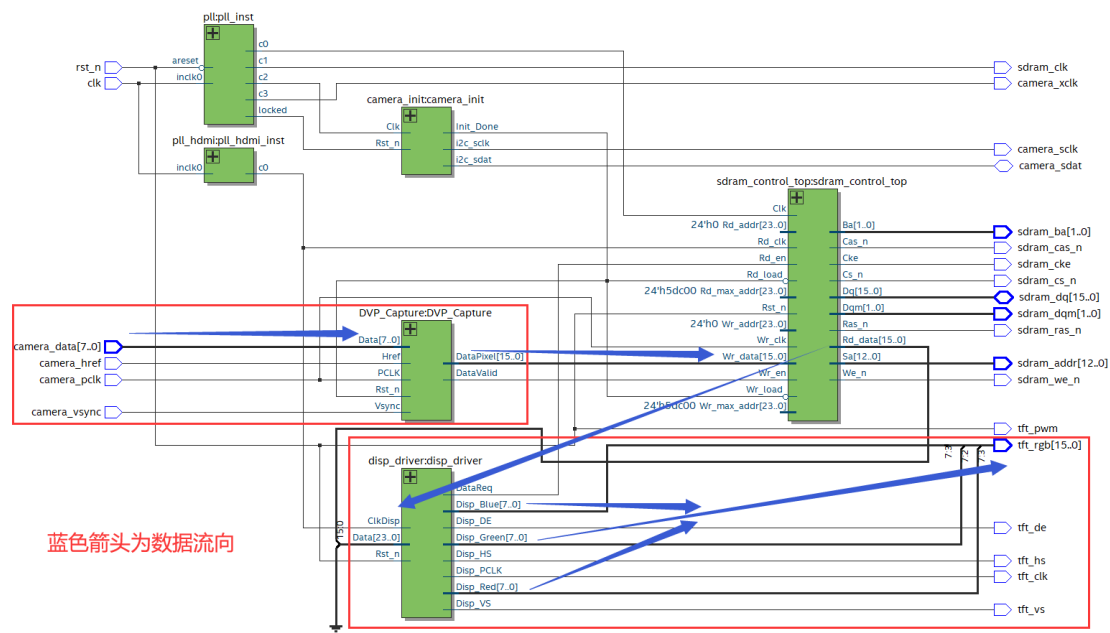


图 1.1-14 系统 RTL 输入原理图总框图

以 RTL 原理图角度验证了整个设计符合预期后，接下来进行板级调试与验证。

### 1.1.6 板级调试

在完成了所有的工程源码设计工作，并成功的得到与原始设计相符的 RTL 设计输出后，接下来进行板级调试和验证。

#### 1.1.6.1 系统所需硬件

1. AC620 开发板 x1
2. 电源线一根

店铺: <https://xiaomeige.taobao.com>

技术博客: <http://www.cnblogs.com/xiaomeige/>

官方网站: [www.corecourse.cn](http://www.corecourse.cn)

技术群组:

3. OV5640 摄像头一个
4. 5 寸 800\*480 分辨率显示屏一块
5. Type-B 线一根（方口线）

### 1.1.6.2 硬件连接图

按照前面介绍的内容，小梅哥团队出品的 AC620 开发板能够很好的兼容 OV5640 摄像头作为输入，5 寸 TFT 显示屏作为输出。实验前，连接好开发板电源线和下载器后，按如下接线方式插接好摄像头和 TFT 屏。

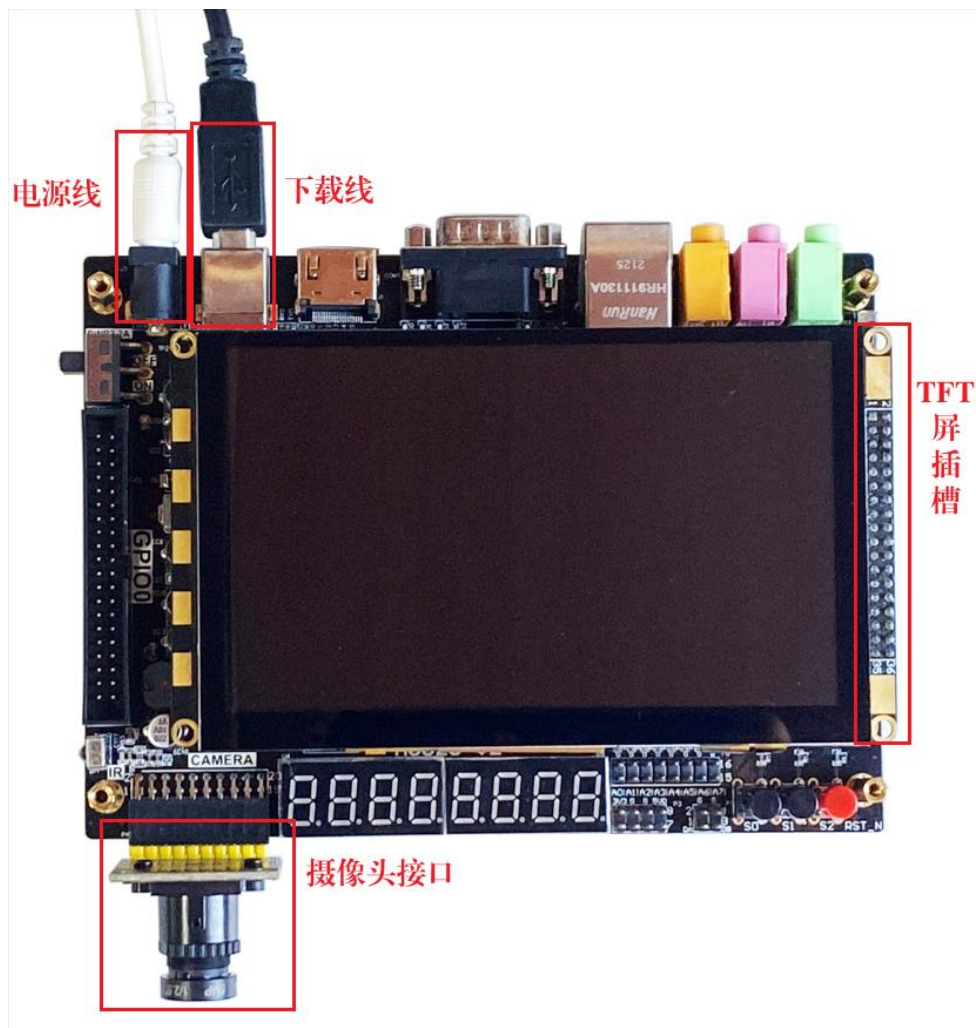


图 1.1-15 OV5640 图像采集 TFT 显示硬件连接图

店铺: <https://xiaomeige.taobao.com>

官方网站: [www.corecourse.cn](http://www.corecourse.cn)

技术博客: <http://www.cnblogs.com/xiaomeige/>

技术群组:

### 1.1.6.3 管脚绑定表

连接好硬件实物后，即可对设计好的工程源码进行管脚绑定。

表 1.1-4 管脚绑定表

| 信号名            | 方向     | 管脚绑定    | 信号名            | 方向     | 管脚绑定    |
|----------------|--------|---------|----------------|--------|---------|
| clk            | Input  | PIN_E1  | sdram_addr[12] | Output | PIN_N11 |
| rst_n          | Input  | PIN_E16 | sdram_addr[11] | Output | PIN_R13 |
| tft_rgb[15]    | Output | PIN_P16 | sdram_addr[10] | Output | PIN_M10 |
| tft_rgb[14]    | Output | PIN_N15 | sdram_addr[9]  | Output | PIN_T14 |
| tft_rgb[13]    | Output | PIN_R16 | sdram_addr[8]  | Output | PIN_R14 |
| tft_rgb[12]    | Output | PIN_P15 | sdram_addr[7]  | Output | PIN_T15 |
| tft_rgb[11]    | Output | PIN_N13 | sdram_addr[6]  | Output | PIN_L11 |
| tft_rgb[10]    | Output | PIN_L12 | sdram_addr[5]  | Output | PIN_M11 |
| tft_rgb[9]     | Output | PIN_K12 | sdram_addr[4]  | Output | PIN_N12 |
| tft_rgb[8]     | Output | PIN_L13 | sdram_addr[3]  | Output | PIN_T13 |
| tft_rgb[7]     | Output | PIN_M12 | sdram_addr[2]  | Output | PIN_P14 |
| tft_rgb[6]     | Output | PIN_L14 | sdram_addr[1]  | Output | PIN_L10 |
| tft_rgb[5]     | Output | PIN_N16 | sdram_addr[0]  | Output | PIN_P11 |
| tft_rgb[4]     | Output | PIN_J16 | sdram_ba[1]    | Output | PIN_M9  |
| tft_rgb[3]     | Output | PIN_K15 | sdram_ba[0]    | Output | PIN_T12 |
| tft_rgb[2]     | Output | PIN_K16 | sdram_cas_n    | Output | PIN_R11 |
| tft_rgb[1]     | Output | PIN_J13 | sdram_cs_n     | Output | PIN_R12 |
| tft_rgb[0]     | Output | PIN_L15 | sdram_ras_n    | Output | PIN_N9  |
| tft_clk        | Output | PIN_J15 | sdram_cke      | Output | PIN_T11 |
| tft_de         | Output | PIN_J11 | sdram_clk      | Output | PIN_T10 |
| tft_hs         | Output | PIN_K11 | sdram_dq[15]   | Bidir  | PIN_P9  |
| tft_pwm        | Output | PIN_J12 | sdram_dq[14]   | Bidir  | PIN_N8  |
| tft_vs         | Output | PIN_J14 | sdram_dq[13]   | Bidir  | PIN_M8  |
| camera_sdat    | Bidir  | PIN_R7  | sdram_dq[12]   | Bidir  | PIN_L8  |
| camera_data[7] | Input  | PIN_P3  | sdram_dq[11]   | Bidir  | PIN_K8  |
| camera_data[6] | Input  | PIN_L7  | sdram_dq[10]   | Bidir  | PIN_L9  |
| camera_data[5] | Input  | PIN_P6  | sdram_dq[9]    | Bidir  | PIN_K9  |
| camera_data[4] | Input  | PIN_N6  | sdram_dq[8]    | Bidir  | PIN_R9  |
| camera_data[3] | Input  | PIN_T6  | sdram_dq[7]    | Bidir  | PIN_R8  |
| camera_data[2] | Input  | PIN_M7  | sdram_dq[6]    | Bidir  | PIN_R6  |
| camera_data[1] | Input  | PIN_P8  | sdram_dq[5]    | Bidir  | PIN_T5  |
| camera_data[0] | Input  | PIN_K10 | sdram_dq[4]    | Bidir  | PIN_R5  |
| camera_href    | Input  | PIN_N3  | sdram_dq[3]    | Bidir  | PIN_T4  |
| camera_pclk    | Input  | PIN_M2  | sdram_dq[2]    | Bidir  | PIN_R4  |
| camera_sclk    | Output | PIN_T2  | sdram_dq[1]    | Bidir  | PIN_T3  |
| camera_vsync   | Input  | PIN_M6  | sdram_dq[0]    | Bidir  | PIN_R3  |
| camera_xclk    | Output | PIN_N5  | sdram_dqm[1]   | Output | PIN_R10 |

店铺: <https://xiaomeige.taobao.com>

官方网站: [www.corecourse.cn](http://www.corecourse.cn)

技术博客: <http://www.cnblogs.com/xiaomeige/>

技术群组:



|  |  |  |              |        |        |
|--|--|--|--------------|--------|--------|
|  |  |  | sdrām_dqm[0] | Output | PIN_T8 |
|  |  |  | sdrām_we_n   | Output | PIN_T9 |

1、如图 1.1-16 所示，在菜单 Assignments 中选择 Pin Planner，也可以直接点击面板上引脚分配的图标；

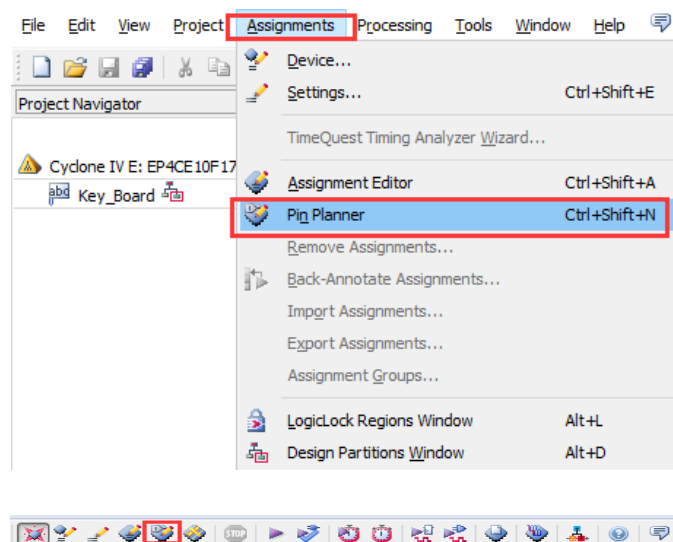


图 1.1-16 进入引脚分配界面选项

2、进入引脚分配的界面之后，按照上面给出的 OV5640 摄像头、TFT 与 AC620 板卡的连接关系以及引脚分配信息，完成引脚分配工作。下图为管脚绑定后的部分效果图。



|     |                |        |         |   |       |         |              |
|-----|----------------|--------|---------|---|-------|---------|--------------|
| in  | camera_data[7] | Input  | PIN_P3  | 3 | B3_N0 | PIN_P3  | 3.3-V LVTTTL |
| in  | camera_data[6] | Input  | PIN_L7  | 3 | B3_N0 | PIN_L7  | 3.3-V LVTTTL |
| in  | camera_data[5] | Input  | PIN_P6  | 3 | B3_N0 | PIN_P6  | 3.3-V LVTTTL |
| in  | camera_data[4] | Input  | PIN_N6  | 3 | B3_N0 | PIN_N6  | 3.3-V LVTTTL |
| in  | camera_data[3] | Input  | PIN_T6  | 3 | B3_N0 | PIN_T6  | 3.3-V LVTTTL |
| in  | camera_data[2] | Input  | PIN_M7  | 3 | B3_N0 | PIN_M7  | 3.3-V LVTTTL |
| in  | camera_data[1] | Input  | PIN_P8  | 3 | B3_N0 | PIN_P8  | 3.3-V LVTTTL |
| in  | camera_data[0] | Input  | PIN_K10 | 4 | B4_N0 | PIN_K10 | 3.3-V LVTTTL |
| in  | camera_href    | Input  | PIN_N3  | 3 | B3_N0 | PIN_N3  | 3.3-V LVTTTL |
| in  | camera_pclk    | Input  | PIN_M2  | 2 | B2_N0 | PIN_M2  | 3.3-V LVTTTL |
| out | camera_pwn     | Output | PIN_P1  | 2 | B2_N0 | PIN_P1  | 3.3-V LVTTTL |
| out | camera_rst_n   | Output | PIN_T7  | 3 | B3_N0 | PIN_T7  | 3.3-V LVTTTL |
| out | camera_sclk    | Output | PIN_T2  | 3 | B3_N0 | PIN_T2  | 3.3-V LVTTTL |
| io  | camera_sdat    | Bidir  | PIN_R7  | 3 | B3_N0 | PIN_R7  | 3.3-V LVTTTL |
| in  | camera_vsync   | Input  | PIN_M6  | 3 | B3_N0 | PIN_M6  | 3.3-V LVTTTL |
| out | camera_xclk    | Output | PIN_N5  | 3 | B3_N0 | PIN_N5  | 3.3-V LVTTTL |
| in  | clk            | Input  | PIN_E1  | 1 | B1_N0 | PIN_E1  | 3.3-V LVTTTL |
| out | led[1]         | Output | PIN_A2  | 8 | B8_N0 | PIN_A2  | 3.3-V LVTTTL |
| out | led[0]         | Output | PIN_B3  | 8 | B8_N0 | PIN_B3  | 3.3-V LVTTTL |
| in  | rst_n          | Input  | PIN_E16 | 6 | B6_N0 | PIN_E16 | 3.3-V LVTTTL |
| out | sdram_addr[12] | Output | PIN_N11 | 4 | B4_N0 | PIN_N11 | 3.3-V LVTTTL |
| out | sdram_addr[11] | Output | PIN_R13 | 4 | B4_N0 | PIN_R13 | 3.3-V LVTTTL |
| out | sdram_addr[10] | Output | PIN_M10 | 4 | B4_N0 | PIN_M10 | 3.3-V LVTTTL |
| out | sdram_addr[9]  | Output | PIN_T14 | 4 | B4_N0 | PIN_T14 | 3.3-V LVTTTL |
| out | sdram_addr[8]  | Output | PIN_R14 | 4 | B4_N0 | PIN_R14 | 3.3-V LVTTTL |
| out | sdram_addr[7]  | Output | PIN_T15 | 4 | B4_N0 | PIN_T15 | 3.3-V LVTTTL |
| out | sdram_addr[6]  | Output | PIN_L11 | 4 | B4_N0 | PIN_L11 | 3.3-V LVTTTL |
| out | sdram_addr[5]  | Output | PIN_M11 | 4 | B4_N0 | PIN_M11 | 3.3-V LVTTTL |
| out | sdram_addr[4]  | Output | PIN_N12 | 4 | B4_N0 | PIN_N12 | 3.3-V LVTTTL |
| out | sdram_addr[3]  | Output | PIN_T13 | 4 | B4_N0 | PIN_T13 | 3.3-V LVTTTL |
| out | sdram_addr[2]  | Output | PIN_P14 | 4 | B4_N0 | PIN_P14 | 3.3-V LVTTTL |
| out | sdram_addr[1]  | Output | PIN_L10 | 4 | B4_N0 | PIN_L10 | 3.3-V LVTTTL |
| out | sdram_addr[0]  | Output | PIN_P11 | 4 | B4_N0 | PIN_P11 | 3.3-V LVTTTL |
| out | sdram_ba[1]    | Output | PIN_M9  | 4 | B4_N0 | PIN_M9  | 3.3-V LVTTTL |
| out | sdram_ba[0]    | Output | PIN_T12 | 4 | B4_N0 | PIN_T12 | 3.3-V LVTTTL |
| out | sdram_cas_n    | Output | PIN_R11 | 4 | B4_N0 | PIN_R11 | 3.3-V LVTTTL |
| out | sdram_cke      | Output | PIN_T11 | 4 | B4_N0 | PIN_T11 | 3.3-V LVTTTL |
| out | sdram_clk      | Output | PIN_T10 | 4 | B4_N0 | PIN_T10 | 3.3-V LVTTTL |
| out | sdram_cs_n     | Output | PIN_R12 | 4 | B4_N0 | PIN_R12 | 3.3-V LVTTTL |
| io  | sdram_dq[15]   | Bidir  | PIN_P9  | 4 | B4_N0 | PIN_P9  | 3.3-V LVTTTL |
| io  | sdram_dq[14]   | Bidir  | PIN_N8  | 3 | B3_N0 | PIN_N8  | 3.3-V LVTTTL |
| io  | sdram_dq[13]   | Bidir  | PIN_M8  | 3 | B3_N0 | PIN_M8  | 3.3-V LVTTTL |
| io  | sdram_dq[12]   | Bidir  | PIN_L8  | 3 | B3_N0 | PIN_L8  | 3.3-V LVTTTL |

图 1.1-17 引脚分配(以 AC620 管脚配置作演示)

3、完成管脚绑定后对工程进行分析和综合，最终生成 sof 文件，然后将其下载到开发板。

## 1.1.7 板级验证

图 1.1-18 为使用 AC620 开发板 OV5640 摄像头图像采集 TFT 显示输出的实验效果图。



图 1.1-18 AC620 开发板 OV5640 摄像头图像采集 TFT 输出显示效果

这样，我们的逻辑设计在板级验证环节就得到了验证。

### 1.1.8 本工程实现 VGA 接口显示的补充说明

前面的文档已经说过，对于本工程，实现 VGA 显示和实现 TFT 显示，原理是相同的，硬件连接上只需要在插接 TFT 显示屏的 GPIO 接口插接 VGA 显示模块即可。从工程源码上来讲，VGA 接口需要对如下参数进行设置修改：

- 1、修改参数表的宏定义 RGB565 换成 RGB888，修改分辨率为 640x480；

文件位置：disp\_parameter\_cfg.v

```
`define HW_VGA

//以下两行预定义根据实际使用的模式，选择一个使能，另外一个使用注释的方式屏蔽
`define MODE_RGB888
//`define MODE_RGB565

//以下 7 行预定义根据实际使用的分辨率，选择一个使能，另外 6 个使用注释的方式屏蔽
```

```
`ifdef HW_TFT43 //使用 4.3 寸 480*272 分辨率显示屏
    `define Resolution_480x272 1 //时钟为 9MHz

`elsif HW_TFT50 //使用 5 寸 800*480 分辨率显示屏
    `define Resolution_800x480 1 //时钟为 33MHz

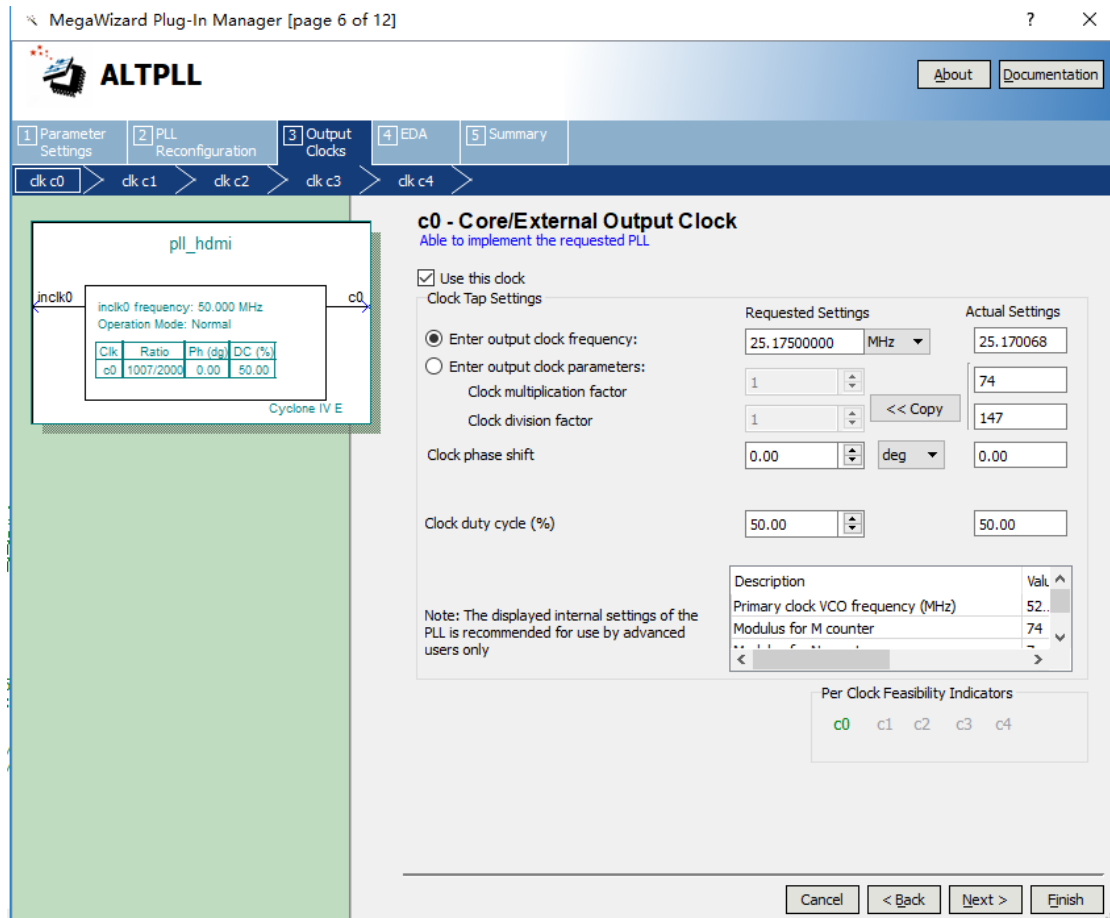
`elsif HW_VGA //使用 VGA 显示器，默认为 640*480 分辨率，也可选择其他分辨率，不过摄像头捕获存储部分当前没做其他分辨率适配
    `define Resolution_640x480 1 //时钟为 25.175MHz
    //`define Resolution_800x480 1 //时钟为 33MHz
    //`define Resolution_800x600 1 //时钟为 40MHz
    //`define Resolution_1024x600 1 //时钟为 51MHz
    //`define Resolution_1024x768 1 //时钟为 65MHz
    //`define Resolution_1280x720 1 //时钟为 74.25MHz
    //`define Resolution_1920x1080 1 //时钟为 148.5MHz
`endif
```

2、修改顶层分辨率设置 IMAGE\_WIDTH 和 IMAGE\_HEIGHT;

文件位置：ov5640\_sdram\_vga

```
parameter IMAGE_WIDTH = 640; //图片宽度
parameter IMAGE_HEIGHT = 480; //图片高度(≤720)
```

3、修改锁相环的频率;



4、修改管脚绑定表。将 VGA 模块的管脚和信号名称对应。具体操作方法为：查询 AC620 的管脚绑定表，然后做出相应的变更。

### 1.1.9 本工程实现 DVI 发送器 HDMI 接口显示的补充说明

在专门介绍 HDMI 接口的章节，我们已经介绍了 TFT 和 HDMI 的相同和不同点。为了让工程能够通过 HDMI 接口进行显示，只需在工程中添加相应的 DVI 接口模块，引出接口，并做出相应的管脚绑定即可。

这里，为了介绍知识的连贯性，我们将 HDMI 显示部分的相关介绍再进行引用如下：

店铺：<https://xiaomeige.taobao.com>

官方网站：[www.corecourse.cn](http://www.corecourse.cn)

技术博客：<http://www.cnblogs.com/xiaomeige/>

技术群组：

关于该模块首先需要进行一点说明，HDMI 兼容 DVI，这里我们使用的是 HDMI 发送的功能，但实际只能算 DVI，因为我们没有加入 HDMI 所拥有的音频数据传输功能。而且受 FPGA 的 IO 口翻转速度限制也仅能实现 DVI 规范所规定的传输速率，达不到 HDMI 规范的高速传输能力。因而我们称之为 DVI 发送器，但实际上我们使用的为 HDMI 接口，介绍协议时多以 DVI 描述，介绍接口时则多以 HDMI 描述。

该模块负责将数据转换为 TMDS 编码的方式进行高速串行传输，该数据发送器仅仅是改变了数据的传输方式，将原本的并行数据转换为高速串行数据输出，以符合 HDMI 协议标准，使用 HDMI 方式传输的过程中，数据的内容没有发生任何的变化。图形处理器发送的是什么数据，最终送到 TFT 驱动器的就是什么数据，不会有哪怕一位数据发生变化。所以图像还原度很高，而且由于 HDMI 传输方式为差分传输，其抵抗干扰的能力也比模拟信号高很多，一般不会被受到干扰。该模块与对应信号关系如下图所示：

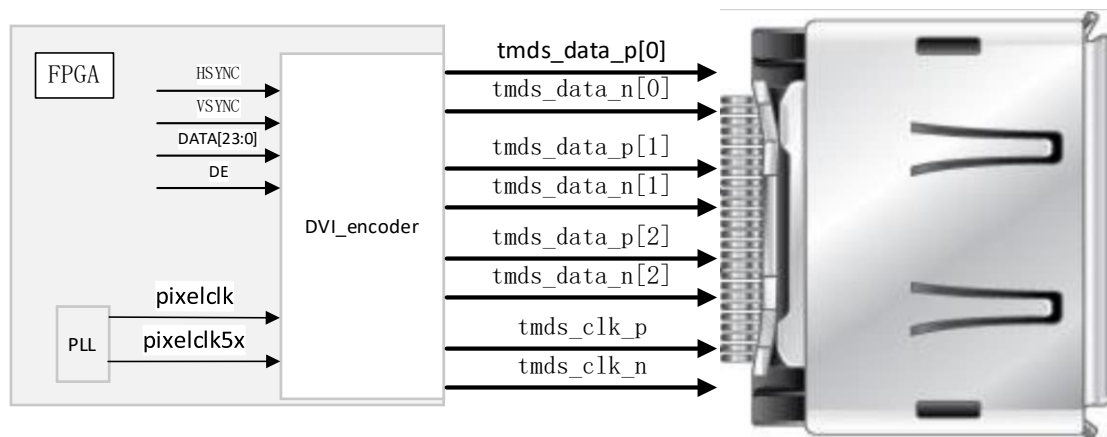


图 1.1-19 HDMI 接口

一个完整的图像传输 TMDS 模块包含三个相同的编码和发送模块，每个 TMDS 模块的结构如下图所示。每个发送模块包含 8 位的像素数据，对应 24 位像素数据中单个颜色的 8 位数据。2 个控制信号 C0、C1，这两个控制信号可以

分别接行同步 (HSYNC)、场同步 (VSYNC) 信号，也可以空置接 0。另外还有一个数据有效信号 DE，该信号用来区分控制数据和像素数据。当 DE 信号为高电平的时候，表明当前数据有效，编码器对 8 位的 Data 数据进行编码。当 DE 信号为低电平的时候，则对 2 位的控制信号进行编码。

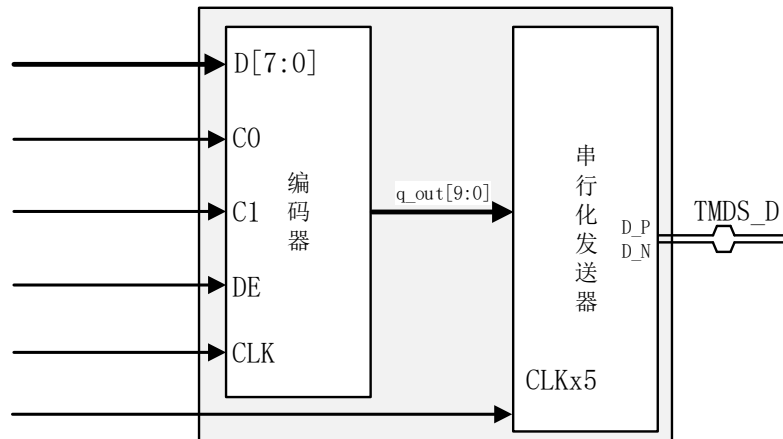


图 1.1-20 TMD5 编码

这里，串行发送器使用的时钟信号 (pixelclk5x) 频率为编码器的 5 倍。至于为什么是 5 倍，这是因为串行发送器在发送数据时候是采用双数据速率的方式传输数据的，一个时钟周期可以传输 2 位信号，所以只需要 5 倍的编码器的时钟即可完成 10 位数据的即时传输。

数据在经过 HDMI 线传输到显示器后，HDMI 显示器内的数据接收器会将 HDMI 接口上的高速串行数字信号转换为并行数据，随即提供给显示器器使用，进而实现图像在 HDMI 显示器上的显示。

本次实验是对以往知识点的一次综合，对应模块的功能及其工作原理在材料文档中都有十分细致的讲解与说明，在学习时可以将对应模块代码与相应的章节资料相互结合印证。

下面，给出本工程改造为 DVI 发送器发送后的代码调整及介绍：

1. 在进行 TFT 工程转 HDMI 工程时，首先将顶层 TFT 显示部分的管脚替换为 tmds 管脚，同时需查阅 AC620 开发板管脚绑定表进行相应的管脚绑定。

```
module ov5640_sdram_hdmi(  
    input          clk,  
    input          rst_n,  
    ... ..  
    //hdmi output  
    output         tmds_clk_p,  
    output         tmds_clk_n,  
    output [2:0]   tmds_data_p,    //rgb  
    output [2:0]   tmds_data_n    //rgb  
);
```

2. 调整锁相环的输出频率、图片的宽度和高度。例如，我们将 HDMI 接口的输出图片分辨率设定为 1280\*720，根据计算，锁相环需提供的 HDMI 频率和 HDMI5 倍频率分别为 74.25Mhz 和 371.25Mhz。

```
wire clk_vga;//74.25m  
wire clk_vgax5;//371.25m  
pll_hdmi pll_hdmi_inst(  
    .inclk0(clk),  
    .c0(clk_vga),  
    .c1(clk_vgax5)  
);  
  
localparam RGB = 0;  
localparam JPEG = 1;  
  
parameter IMAGE_WIDTH = 1280; //图片宽度  
parameter IMAGE_HEIGHT = 720; //图片高度(≤720)  
parameter IMAGE_FLIP = 0; //0: 不翻转, 1: 上下翻转  
parameter IMAGE_MIRROR = 0; //0: 不镜像, 1: 左右镜像
```

下面是锁相环的输出频率配置修改情况：

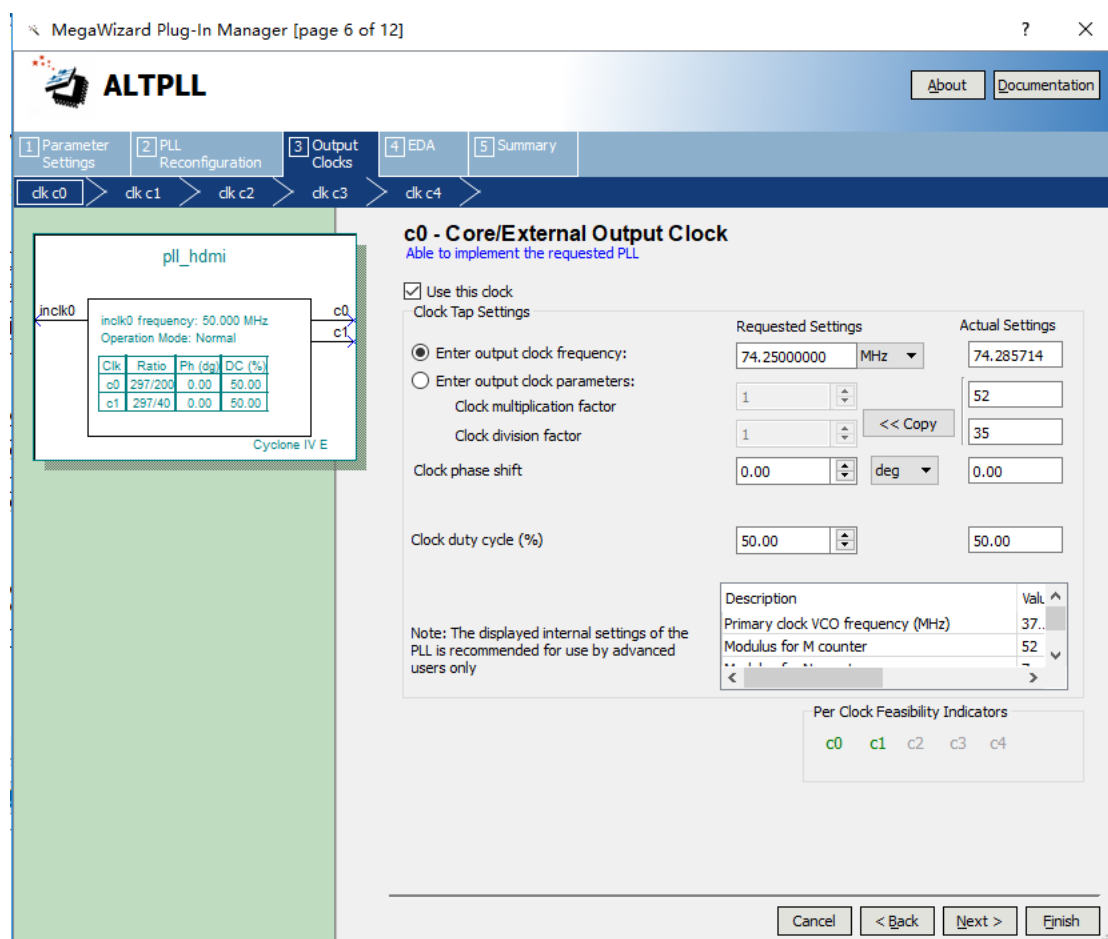


图 1.1-21 pll\_hdmi 锁相环输出配置 1



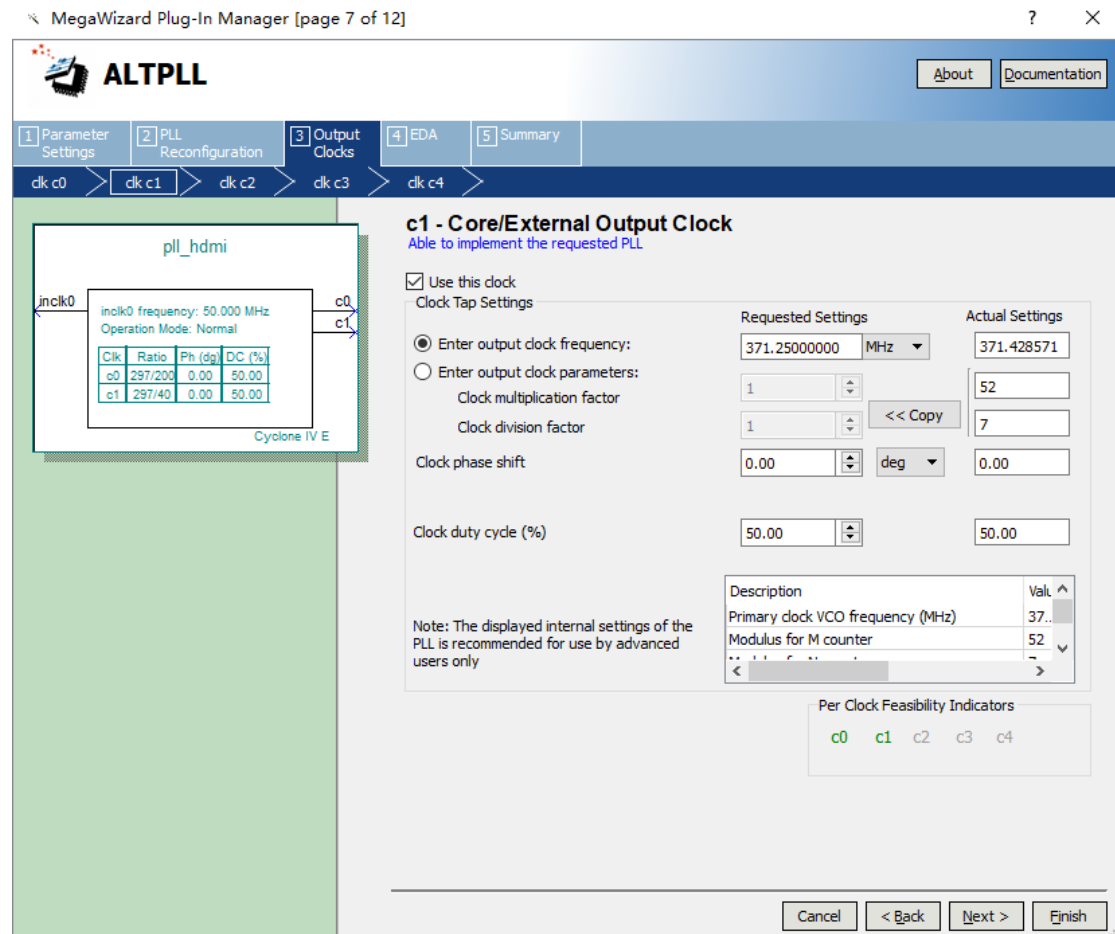


图 1.1-22 pll\_hdmi 锁相环输出配置 2

3. 此外，将 disp\_driver 的输出信号，输送给 dvi 控制器即可。

```
disp_driver disp_driver(
    .ClkDisp(clk_vga),
    .Rst_n(rst_n),
    .Data({RGB_DATA[15:11], 3'd0, RGB_DATA[10:5], 2'd0, RGB_DATA[4:0], 3'd0}),
    .DataReq(DataReq),
    .H_Addr(H_Addr),
    .V_Addr(V_Addr),
    .Disp_HS(video_hs),
    .Disp_VS(video_vs),
    .Disp_Red(video_r),
    .Disp_Green(video_g),
    .Disp_Blue(video_b),
    .Disp_DE(video_de),
```

```
        .Disp_PCLK()  
    );  
  
    dvi_encoder u_dvi_encoder(  
        .pixelclk(clk_vga),           // system clock  
        .pixelclk5x(clk_vga5),       // system clock x5  
        .rst_n(rst_n),               // reset  
        .blue_din(video_b),          // Blue data in  
        .green_din(video_g),         // Green data in  
        .red_din(video_r),           // Red data in  
        .hsync(video_hs),            // hsync data  
        .vsync(video_vs),            // vsync data  
        .de(video_de),               // data enable  
        .tmds_clk_p(tmds_clk_p),  
        .tmds_clk_n(tmds_clk_n),  
        .tmds_data_p(tmds_data_p), //rgb  
        .tmds_data_n(tmds_data_n) //rgb  
    );
```

这样，就完成了 TFT 工程到 HDMI 工程的移植。这里，也提供了 AC620 开发板的移植 HDMI 后工程。

### 1.1.10 常见问题说明

1. 由于本工程需要用到 SDRAM 缓存，而 SDRAM 缓存需要一组读写 FIFO 配合完成数据的缓存工作，因此，在有效数据写入前，需要将读写 FIFO 清空，以保证缓存的数据都为有效的像素数据。
2. 本工程设计的 disp\_driver 模块可兼容 RGB888 数据格式和 RGB565 数据格式，如果使用 RGB565 数据格式，使用三元色的适配位宽进行位截取和位拼接即可。